网络空间安全创新创业实践

Merkle Tree Project

实验报告

王祥宇---202000460053

项目时间：2022 年 7 月 13 日

目录

一、项目任务

Merkle Tree 项目在实现 Merkle Tree 的基础上附有三个小任务，其中最后两个存在证明和不存在证明可以合成一个任务。如下图：

*Project: Impl Merkle Tree following RFC6962
- Construct a Merkle tree with 10w leaf nodes
- Build inclusion proof for specified element
- Build exclusion proof for specified element

二、实验过程及思路

任务一：Implement Merkle Tree following RFC6962

完成该项目首先需要实现 Merkle Tree，所以将其列为任务一。关于 Merkle Tree 的存储采用二维列表的形式，其中每一层节点构成一个一维列表，并且每个一维列表按照从下往上（即从叶子节点到根节点）的顺序进行存储。用该种存储思路可以方便的将 Merkle Tree 中的父子节点关系给表示出来，并且在这样的表示下，二维列表的长度即为树的深度，二维列表中第一个元素即为叶子节点。树中的 Hash 函数采用最经典的 sha256 算法。具体实现步骤如下所示：

Step-1：前期准备

声明 Merkle Tree 二位列表并根据叶子节点数目计算深度。

Step-2：叶子层的实现

计算每个叶子节点（即数据）的 hash 值，将每个 hash 值放入 Merkle Tree

二位列表的第一个一维列表，从而完成叶子结点的实现。第一二步的代码实现如下图：

```
MerkleTree = [[]]
length = len(DATA)
Depth = math.ceil(math.log(length, 2))+1
MerkleTree[0] = [(hashlib.sha256(i.encode())).hexdigest() for i in DATA]
```

Step-3：从叶子层依次向上实现

对于每两个子节点，计算其和的 hash 值作为这两个叶子结点的父节点并写入 Merkle Tree 列表中的对应位置。对于该层的其他节点依次执行该步骤即可。

对于最后落单的节点（即该层有奇数个节点，最后的节点即为落单的节点），将其直接往上一层，相当于自己作为自己的父节点即可。然后对所有层均进行如上操作即可。相关代码如下图所示：

```
for i in range(1, Depth):
    p = math.floor(len(MerkleTree[i-1])/2)
    MerkleTree.append([(hashlib.sha256(MerkleTree[i-1][2*j].encode() + \
                        MerkleTree[i-1][2*j+1].encode())).hexdigest() for j in range(0, p)])
    if len(MerkleTree[i-1])%2 == 1:
        MerkleTree[i].append(MerkleTree[i-1][-1])
```

Step-4：任务测试

输入 7 个数据以此为 a~g，调用上述过程的函数。代码实现如下图：

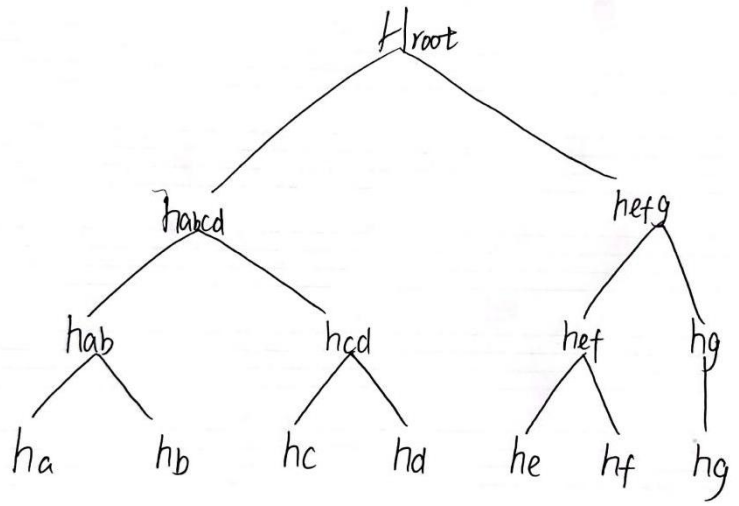
```
#任务一、Create a MerkleTree
DATA = ['a','b','c','d','e','f','g']
Tree_1 = CreatTree(DATA)
print('一、Create a MerkleTree\n')
print(Tree_1)
print('\n')
```

测试结果如下：

```
一、Create a MerkleTree

[['ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb', '3e23e8160
039594a33894f6564e1b1348bbd7a0088d42c4acb73eeaed59c009d', '2e7d2c03a9507ae265ecf
5b5356885a53393a2029d241394997265a1a25aefc6', '18ac3e7343f016890c510e93f93526116
9d9e3f565436429830faf0934f4f8e4', '3f79bb7b435b05321651daefd374cdc681dc06faa65e3
74e38337b88ca046dea', '252f10c83610ebca1a059c0bae8255eba2f95be4d1d7bcfa89d7248a8
2d9f111', 'cd0aa9856147b6c5b4ff2b7dfee5da20aa38253099ef1b4a64aced233c9afe29'], [
'62af5c3cb8da3e4f25061e829ebeea5c7513c54949115b1acc225930a90154da', 'd3a0f1c792c
cf7f1708d5422696263e35755a86917ea76ef9242bd4a8cf4891a', '1b3dae70b4b0a8fd252a787
9ec67283c0176729bfebc51364fb9e9fb0598ba9e', 'cd0aa9856147b6c5b4ff2b7dfee5da20aa3
8253099ef1b4a64aced233c9afe29'], ['58c89d709329eb37285837b042ab6ff72c7c8f74de044
6b091b6a0131c102cfd', 'cba73b48ac8cc7729cd5b96ad523150a94cf06de0796ccd075199ed68
2657932'], ['d4eaf58f7485ba6e770c5825d39c4b7ffd4d74ed2ee036c7d7804763c77d0e3f']]
```

该测试将相当于如下的 Merkle Tree 逻辑图：



所以我们可以看到最后的结果中一维列表的长度（即元素个数）依次为：7，4，2，1。符合逻辑 Merkle Tree 图。

任务二：Construct a Merkle tree with 10w leaf nodes

任务二的实现在任务一的基础上变得很简单，只需要将 10w 个消息（数据）调用任务一中的 Create 函数来生成 Merkle Tree 即可。

Step-1：生成 10w 个消息

首先使用 python 中的 random.sample 函数生成随机的 10w 个消息，这里为了生成消息的便利将每个消息的长度定为 5，且仅为字母或数字。相关

代码如下图：

```
#2. Generate 10w messages
def GenerateMessages():
    return [''.join(random.sample('abcdefghijklmnopqrstuvwxyz0123456789',\
                                  5)) for i in range(0,N)]
```

Step-2：生成 Merkle Tree

在生成 10w 个消息后，调用任务一中的 Create 函数即可生成 Merkle Tree。

Step-3：任务测试

调用 Step-1 中的 GenerateMessages 函数来生成消息，在调用 Create 即可。

```
#任务二、Construct a Merkle Tree with 10w leaf nodes
TestMessages = GenerateMessages()
Tree_2 = Create(TestMessages)
print('二、Construct a Merkle Tree with 10w leaf nodes\n')
print(Tree_2)
print('\n')
```

由于有 10w 个数据，数目太大，所以该任务运行时间较长，故在此不展示测试的运行结果。

任务三：Build inclusion or exclusion proof for specified element

任务三的证明分为两个阶段，首先根据消息 m 的 hash 值生成一个 Evidence 数组，用来作为 m 存在于 Merkle Tree 中的证据。然后去验证 Evidence 是否符合 Merkle Tree 的逻辑。即：如果 Evidence 数组也是一个 Merkle Tree（即是一个二维数组，并且父节点等于子节点之和），就说明给出的 Evidence 数组是正确的，即说明消息 m 在指定的 Merkle Tree 中。

Step-1：找出消息 m 对应的索引

对于消息 m，根据其 hash 值利用 index 函数得出其索引。注意，在 python 语言中是从 0 开始的，实现代码如下图所示：

```
h = (hashlib.sha256(m.encode())).hexdigest()

try:
    n=Tree[0].index(h)
except:
    print("The leaf node is not in the tree!!!")
```

Step-2：根据索引 n 的奇偶性分情况讨论，构造 Evidence 数组

如果 n 为奇数节点（注意，由于 python 下标是从 0 开始的，所以在代码实现中的奇偶性判断和此处的语言描述正好相反），此时要判断是否为该层的最后一个节点，如果是的话，说明该节点不在该层中，可能会在上一层，所以在此处直接跳过；如果不是最后一个节点，则将其和其右兄弟节点一起加入到 Evidence 数组中。

如果 n 为偶数节点，则直接将其和其兄弟左节点加入到 Evidence 数组中。

这样从叶子层一直到根节点的下一层，最后将根节点单独加入到 Evidence 中即可。实现代码如下图所示：

```
for d in range(0,Depth):
    if n%2==0:
        if n == len(Tree[d]) - 1:
            pass
        else:
            Evidence.append([Tree[d][n],Tree[d][n+1]])

    else:
        Evidence.append([Tree[d][n-1], Tree[d][n]])

    n = math.floor(n/2)

Evidence.append([Tree[-1][0]])
```

Step-3：验证 Evidence 是否符合 Merkle Tree 的逻辑结构，从而说明消息 m 是否在指定的 Merkle Tree 中

首先要得到消息 m 的 hash 值。然后对 Evidence 数组进行基本的检查，比如检查根节点是否正确，消息 m 的 hash 值是否在 Evidence 中等等。实现代码如下图：

```
h = (hashlib.sha256(m.encode())).hexdigest()
if h != Evidence[0][0] and h != Evidence[0][1]:
    return False

if Evidence[-1][0] != Top:
    return False
```

然后验证 Evidence。将每一层的节点相加，检查是否等于其父节点。如果每一层都符合，说明 Evidence 符合 Merkle Tree 的逻辑结构，则说明消息 m 位于指定的 Merkle Tree 中。相反，如果不符合，则证明其不在 Merkle Tree 中。实现代码如下图：

```
Depth = len(Evidence)
for i in range(0,Depth-1):
    node = (hashlib.sha256(Evidence[i][0].encode() + Evidence\
                           [i][1].encode())).hexdigest()
    if node != Evidence[i+1][0] and node != Evidence[i+1][1]:
        return False

if (hashlib.sha256(Evidence[-2][0].encode() + Evidence[-2][1].\
                   encode())).hexdigest() != Evidence[-1][0]:
    return False

return True
```

Step-4：inclusion proof 测试

存在证明使用任务二中的 10w 个消息中的随机一个进行验证。为了方便展示结果，将任务二中的 10w 换成 7，即生成 7 条消息的 Merkle Tree，然后在这 7 条消息中随机选一条消息进行验证。结果肯定是 True。实现代码如下图：

```
#任务三、Build inclusion proof for specified element
print('三、Build inclusion proof for specified element')
n=random.randint(0,N-1) #Choose a random message
Evidence = GenerateEvidence(TestMessages[n],Tree_2)
print("The evidence is:")
print(VerifyEvidence(TestMessages[n],Evidence,Tree_2[-1][0]))
print('\n')
```

测试结果如下图：

```
三、Build inclusion proof for specified element
The evidence is:
True
```

Step-5：exclusion proof 测试

不存在证明使用任务一中的 Merkle Tree 进行测试。测试消息"k"是否在该树中。结果应该输出"The leaf node is not in the tree!!!"并且会报错。实现代码如下图：

```
#任务四、Build exclusion proof for specified element
print('四、Build exclusion proof for specified element')
Evidence=GenerateEvidence("k",Tree_1)
print("The evidence is:")
print(VerifyEvidence("k",Evidence,Tree_1[-1][0]))
```

测试结果如下图：

```
四、Build exclusion proof for specified element
The leaf node is not in the tree!!!
Traceback (most recent call last):
  File "D:\桌面\网安创新创业\王祥宇-202000460053
```

三、实验反思与总结

通过此项目的三个任务，了解并实现了 Merkle Tree 的相关内容，对 Merkle Tree 有了更加深入的了解。也了解到了 Merkle Tree 在 bitcoin 和区块链中的应

用等等，熟悉了相关密码操作在 python 的实现方法。为以后的密码学习和密码工程的实现奠定了基础。